Subject: styling of widgets (animation / look and feel)
Posted by dodobar on Sun, 16 Apr 2023 18:54:50 GMT
View Forum Message <> Reply to Message

I'm reflecting on challenges of UI design, coming from this sector its tricky to find libraries allowing for easy widget styling and animation trends.
it's no easy task as it is a fundamental to the underlying architecture , but interesting to know how people feel about this in U++:
looking briefly at the code there are several steps perhaps required for implementing this:

Implement hardware acceleration (like OpenGL,Vulkan). (this seems to be some support already with the GLDraw)
Scene graph-like data structure for efficient UI management.
Use retained mode rendering to store and manage UI element states.
Support asynchronous loading of resources and UI elements.
Minimize unnecessary redraws with efficient invalidation and update mechanisms.
Develop a animation and transition system.
Integrating widgets to use this styling, override default behaviours

Subject: Re: styling of widgets (animation / look and feel)
Posted by Novo on Mon, 17 Apr 2023 13:53:00 GMT
View Forum Message <> Reply to Message

hardware acceleration (like OpenGL,Vulkan) requires a tesselation algorithm. This is a biggest problem.

Subject: Re: styling of widgets (animation / look and feel)
Posted by mirek on Tue, 18 Apr 2023 07:17:29 GMT
View Forum Message <> Reply to Message

dodobar wrote on Sun, 16 April 2023 20:54I'm reflecting on challenges of UI design, coming from this sector its tricky to find libraries allowing for easy widget styling and animation trends.
it's no easy task as it is a fundamental to the underlying architecture , but interesting to know how people feel about this in U++:
looking briefly at the code there are several steps perhaps required for implementing this:

I think U++ can be a good starting point for you to experiment, e.g. with VirtualGUI.

OTOH, investing in this does not make much sense for current typical use cases. U++ typical job is gargantuan engineering or bussiness application - applications with hunderds GUI dialogs that do stuff. We are more concerned how to develop / maintain such apps quickly and cheaply than about how fancy they look - as long as they are not TOO ugly.

Subject: Re: styling of widgets (animation / look and feel)
Posted by dodobar on Wed, 19 Apr 2023 22:37:08 GMT
View Forum Message <> Reply to Message

I appreciate your perspective on a primary use cases of U++ .
However, I believe that incorporating simple animation and a rich set of styling abilities can still be beneficial to U++ users, even for those focused on engineering or business applications.

Enhanced user experience:
Simple animation and styling can improve user satisfaction and productivity, making software navigation easier and reducing learning curves.
They want to use your software because it's form and function are in-line.

Attract diverse developers:
A flexible styling system can appeal to a broader range of developers (front end and backend) , expanding the U++ community and encouraging innovation.
modern interfaces are exactly that "modern" and I feel it's important for a framework to offer that, after all it's fundamentally a UI "user interface"

Cheers

Subject: Re: styling of widgets (animation / look and feel)
Posted by mirek on Thu, 20 Apr 2023 06:58:17 GMT
View Forum Message <> Reply to Message

dodobar wrote on Thu, 20 April 2023 00:37I appreciate your perspective on a primary use cases of U++ .
However, I believe that incorporating simple animation and a rich set of styling abilities can still be beneficial to U++ users, even for those focused on engineering or business applications.

Enhanced user experience:
Simple animation and styling can improve user satisfaction and productivity, making software navigation easier and reducing learning curves.
They want to use your software because it's form and function are in-line.

Attract diverse developers:
A flexible styling system can appeal to a broader range of developers (front end and backend) , expanding the U++ community and encouraging innovation.
modern interfaces are exactly that "modern" and I feel it's important for a framework to offer that, after all it's fundamentally a UI "user interface"

Cheers

We already have flexible styling, which admitedly does not support animations yet. I am willing to consider adding animation support there, but it really has to be cheap...

Any suggestions? (but not empty statements, show me the code).

Subject: Re: styling of widgets (animation / look and feel)
Posted by dodobar on Thu, 20 Apr 2023 11:00:33 GMT
View Forum Message <> Reply to Message

my understanding of the fundamental framework is Limited but the general implementation overview could be:

*Base animation class for common functionality.
*Specific animation subclasses (e.g., position, size, color).
*Modify U++ widgets to support animated properties.
*Implement an animation timing system (e.g., timer, loop, generic ease in\out curves).
*Integrate animations with the event system.
*Provide helper functions for common animations and transitions.?

Rough base class examples:

```
#include <Core/Core.h>
#include <CtrlCore/CtrlCore.h>

class BaseAnimation {
public:

    enum EasingCurveType {
        LINEAR,
        EASE_IN_QUAD,
        EASE_OUT_QUAD,
        EASE_IN_OUT_QUAD,
        EASE_IN_CUBIC,
        EASE_OUT_CUBIC,
        EASE_IN_OUT_CUBIC,
        CUSTOM
    };

    BaseAnimation(Ctrl& target, int duration)
        : target_(target), duration_(duration), easing_curve_(LINEAR) {}

    Callback WhenAnimationCompleted;
```

```
    BaseAnimation& Loop(bool loop);
    bool IsLooping() const;

    BaseAnimation& Reverse(bool reverse);
    bool IsReversed() const;

    BaseAnimation& Duration(int duration);
    int GetDuration() const;

    void Start();
    void Pause();
    void Resume();
    void Stop();

    double GetProgress() const;

    BaseAnimation& EasingCurve(EasingCurveType curve);
    BaseAnimation& CustomCurve(const Vector<Pointf>& curve_points);

protected:
    virtual void UpdateProgress(double progress) = 0;
    double ApplyEasingCurve(double progress);

private:
    Ctrl& target_;
    int duration_;
    EasingCurveType easing_curve_;
    Vector<Pointf> custom_curve_points_;
    int timer_;

    void OnTimer();
};
```

Example Position (sorry if the coding style is not 100% U++)

```
#include "BaseAnimation.h"
#include <CtrlCore/ctrl.h> // For Color and Point
#include <CtrlLib/CtrlLib.h> // For Widget class

class PositionAnimation : public BaseAnimation {
public:
    PositionAnimation(Widget& target, const Point& end_position, int duration);

protected:
    virtual void UpdateProgress(double progress) override;
```

```
private:
    Widget& target_;
    Point start_position_;
    Point end_position_;
};

PositionAnimation::PositionAnimation(Widget& target, const Point& end_position, int duration)
    : BaseAnimation(duration), target_(target), end_position_(end_position) {
    start_position_ = target.GetScreenView().TopLeft();
}

void PositionAnimation::UpdateProgress(double progress) {
    Point current_position;
    current_position.x = start_position_.x + (end_position_.x - start_position_.x) * progress;
    current_position.y = start_position_.y + (end_position_.y - start_position_.y) * progress;
    target_.SetRect(target_.GetScreenView().SetTopLeft(current_position));
}
```

Example Colour ?

```
class ColorAnimation : public BaseAnimation {
public:
    ColorAnimation(Widget& target, const Color& end_color, int duration);

protected:
    virtual void UpdateProgress(double progress) override;

private:
    Widget& target_;
    Color start_color_;
    Color end_color_;
};

ColorAnimation::ColorAnimation(Widget& target, const Color& end_color, int duration)
    : BaseAnimation(duration), target_(target), end_color_(end_color) {
    start_color_ = target.GetBackground();
}

void ColorAnimation::UpdateProgress(double progress) {
    Color current_color;
    current_color.r = start_color_.r + (end_color_.r - start_color_.r) * progress;
    current_color.g = start_color_.g + (end_color_.g - start_color_.g) * progress;
    current_color.b = start_color_.b + (end_color_.b - start_color_.b) * progress;
    target_.SetBackground(current_color);
}
```

Not So sure about the integration to the widgets but a approach ?

```
#include "PositionAnimation.h"
#include "ColorAnimation.h"

namespace Upp {

class StaticText : public Ctrl {
public:
    // ... existing class content ...

    StaticText& AnimatePosition(const Point& end_position, int duration);
    StaticText& AnimateColor(const Color& end_color, int duration);
    // ... other animations ?
private:
    void OnAnimationCompleted();
};

} // namespace Upp
```

Example Usage

```
#include <CtrlLib/CtrlLib.h>

using namespace Upp;

GUI_APP_MAIN {
    TopWindow win;
    win.SetRect(0, 0, 800, 600);

    StaticText static_text;
    static_text.SetLabel("Hello, Animated World!");
    static_text.SetRect(10, 10, 200, 50);
    win.Add(static_text);

    win.Run();

    // Example usage of the animation methods
    static_text.AnimatePosition(Point(100, 100), 500); // Move StaticText to (100, 100) in 500 ms
    static_text.AnimateColor(Blue(), 1000); // Change background color to blue in 1000 ms
}
```

Custom example

```
Vector<Pointf> custom_curve_points = {
   {0.0, 0.0},
   {0.4, 0.8},
   {0.6, 0.2},
   {1.0, 1.0}
};

PositionAnimation animation(static_text, Point(100, 100), 500);
animation.CustomCurve(custom_curve_points);
animation.Start();
```

Subject: Re: styling of widgets (animation / look and feel)
Posted by dodobar on Thu, 20 Apr 2023 11:06:16 GMT
View Forum Message <> Reply to Message

Some Generic Curve processing :roll:

```
BaseAnimation& BaseAnimation::EasingCurve(EasingCurve curve) {
   easing_curve_ = curve;
   return *this;
}

//allow for a user style curve
BaseAnimation& BaseAnimation::CustomCurve(const Vector<Pointf>& curve_points) {
   easing_curve_ = CUSTOM;
   custom_curve_points_ = curve_points;
   return *this;
}

double BaseAnimation::ApplyEasingCurve(double progress) {
   switch (easing_curve_) {
      case EASE_IN_QUAD:
         return progress * progress;
      case EASE_OUT_QUAD:
         return progress * (2 - progress);
      case EASE_IN_OUT_QUAD:
         return progress < 0.5 ? 2 * progress * progress : -1 + (4 - 2 * progress) * progress;
      case EASE_IN_CUBIC:
         return progress * progress * progress;
      case EASE_OUT_CUBIC:
```

```
            return (--progress) * progress * progress + 1;
        case EASE_IN_OUT_CUBIC:
            return progress < 0.5 ? 4 * progress * progress * progress : (progress - 1) * (2 * progress -
2) * (2 * progress - 2) + 1;
        case CUSTOM:
            if (custom_curve_points_.IsEmpty()) {
                return progress;
            }

            int index = 0;
            while (index + 1 < custom_curve_points_.GetCount() && progress >
custom_curve_points_[index + 1].x) {
                ++index;
            }

            if (index + 1 < custom_curve_points_.GetCount()) {
                Pointf p1 = custom_curve_points_[index];
                Pointf p2 = custom_curve_points_[index + 1];
                double ratio = (progress - p1.x) / (p2.x - p1.x);
                return p1.y + ratio * (p2.y - p1.y);
            } else {
                return custom_curve_points_.Top().y;
            }
        case LINEAR:
        default:
            return progress;
    }
}
```

Subject: Re: styling of widgets (animation / look and feel)
Posted by mirek on Thu, 20 Apr 2023 12:22:24 GMT
View Forum Message <> Reply to Message

Please check how styling ("Chameleon") in U++ works first. (You can start by searching for ChPaint in the code).

Note that the fundamental goal of Chameleon is to provide the possibility of "host platform consistent" look. Animation has to be integrated within that framework.

Mirek

Subject: Re: styling of widgets (animation / look and feel)
Posted by dodobar on Thu, 20 Apr 2023 17:17:57 GMT
View Forum Message <> Reply to Message

Thanks, however the base classes I provided could still be relevant with a few modifications
To be more in line with what you're suggesting , possibly functions passed to it as the easing

this also would allow for custom using functions I've included a couple of generics.
(but you get the idea)

```
typedef double (*EasingFunction)(double);

struct ChDuration : ChStyle<ChDuration> { int value; };
#define CH_DURATION(name, init) CH_VAR(ChDuration, int, name, init)

struct ChEasing : ChStyle<ChEasing> { EasingFunction value; };

#define CH_EASING(name, init) CH_VAR(ChEasing, EasingFunction, name, init)
double LinearEasing(double t) {
    return t;
}

double EaseInQuad(double t) {
    return t * t;
}

double EaseOutQuad(double t) {
    return t * (2 - t);
}

double EaseInOutQuad(double t) {
    return t < 0.5 ? 2 * t * t : -1 + (4 - 2 * t) * t;
}

double EaseInCubic(double t) {
    return t * t * t;
}

double EaseOutCubic(double t) {
    double f = t - 1;
    return f * f * f + 1;
}

double EaseInOutCubic(double t) {
    return t < 0.5 ? 4 * t * t * t : (t - 1) * (2 * t - 2) * (2 * t - 2) + 1;
}

double EaseInExpo(double t) {
    return (t == 0) ? 0 : pow(2, 10 * (t - 1));
}
```

```
double EaseOutExpo(double t) {
    return (t == 1) ? 1 : 1 - pow(2, -10 * t);
}

double EaseInOutExpo(double t) {
    if (t == 0) return 0;
    if (t == 1) return 1;
    if ((t *= 2) < 1) return 0.5 * pow(2, 10 * (t - 1));
    return 0.5 * (-pow(2, -10 * --t) + 2);
}
```

example usage?

```
CH_DURATION(ButtonAnimationDuration, 300) // Define animation duration for button, e.g., 300ms
CH_EASING(ButtonEasingFunction, EaseInOutQuad) // Define easing function for button animation
```

Im guessing at this point but I assume will need new function definitions to include the new animation parameters:
perhaps a blank function so existing code is not effected (EasingFunction NoEasing)

```
void ChLookFn(Value (*fn)(Draw& w, const Rect& r, const Value& look, int lookop, Color ink, int duration, EasingFunction easing=NoEasing ?? ));
```

Perhaps even defining custom functions?

```
Value MyCustomLookFunction(Draw& w, const Rect& r, const Value& look, int lookop, Color ink, int duration, EasingFunction easing);
```

then example usage ? possibly? (Again these are my estimations is to the real code.)

```
ChLookFn(MyCustomLookFunction, ButtonAnimationDuration(), ButtonEasingFunction());
```

The Base class would look a little different taking the easing functions and providing a simpler set of functions to process animatable style components ?

```
class BaseAnimation {
public:
    BaseAnimation(int duration, EasingFunction easing)
        : duration_(duration), easing_(easing) {}

    double GetProgress(double t) const {
        return easing_(t / duration_);
    }

    Color LerpColor(const Color& start, const Color& end, double progress) const {
        int red = start.GetR() + progress * (end.GetR() - start.GetR());
        int green = start.GetG() + progress * (end.GetG() - start.GetG());
        int blue = start.GetB() + progress * (end.GetB() - start.GetB());
        int alpha = start.GetA() + progress * (end.GetA() - start.GetA());
        return Color(red, green, blue, alpha);
    }

    Point LerpPosition(const Point& start, const Point& end, double progress) const {
        int x = start.x + progress * (end.x - start.x);
        int y = start.y + progress * (end.y - start.y);
        return Point(x, y);
    }

    Size LerpSize(const Size& start, const Size& end, double progress) const {
        int cx = start.cx + progress * (end.cx - start.cx);
        int cy = start.cy + progress * (end.cy - start.cy);
        return Size(cx, cy);
    }

    float LerpFloat(float start, float end, double progress) const {
        return start + progress * (end - start);
    }

    double LerpScale(double start, double end, double progress) const {
        return start + progress * (end - start);
    }

private:
    int duration_;
    EasingFunction easing_;
};
```

example implementation using the CHpaint

```
Color start_color = SColorFace();
Color end_color = SColorHighlight();
```

Color current_color = animation.LerpColor(start_color, end_color);

...

ChPaint(w, r, look, current_color);

...

ChLookFn(MyCustomLookFunction, ButtonAnimationDuration(), ButtonEasingFunction());

Subject: Re: styling of widgets (animation / look and feel)
Posted by mirek on Thu, 20 Apr 2023 19:51:50 GMT
View Forum Message <> Reply to Message

dodobar wrote on Thu, 20 April 2023 19:17Thanks, however the base classes I provided could still be relevant with a few modifications
To be more in line with what you're suggesting , possibly functions passed to it as the easing

Those are trivialities. Obviously you need some time based transition to animate, but that is not the problem.

Problem is that following code:

Quote:

Color start_color = SColorFace();
Color end_color = SColorHighlight();
Color current_color = animation.LerpColor(start_color, end_color);

cannot be code. It has to be parametrised so it can be replaced it with any (or no) animation

Think about Button. Its look (and feel) is parametrized as

```
struct Style : ChStyle<Style> {
  Value look[4];
  Color monocolor[4], textcolor[4];
  Point pressoffset;
  int   focusmargin;
  int   overpaint;
  Font  font;
  Image ok, cancel, exit;
  bool  transparent;
```

```
  bool  focus_use_ok;
};
```

Now buton has 4 basic states (normal, hot (when mouse is over), pushed, disabled). Proper animation support most provide transition beween any 2 of them (that is 12 transitions if I count right) while allowing to set such animation in Style somehow and also while not singificantly complicating Option::Draw. Ideally, animation code, if any, should be completely outside of Option code.

EDIT: Moreover, it needs correct transitions between partial states (e.g. user moves mouse over, so hot state ends before animation is complete).

Mirek

Subject: Re: styling of widgets (animation / look and feel)
Posted by dodobar on Thu, 20 Apr 2023 20:36:33 GMT
View Forum Message <> Reply to Message

agreed this was to get the ball rolling:

perhaps:

```
struct Style : ChStyle<Style> {
    Value look[4];
    Color monocolor[4], textcolor[4];
    Point pressoffset;
    int   focusmargin;
    int   overpaint;
    Font  font;
    Image ok, cancel, exit;
    bool  transparent;
    bool  focus_use_ok;

    // Animation properties
  enum AnimationFinishBehavior {
        GOTO_FIRST,      // Go to the first frame when the animation is interrupted
        GOTO_LAST,       // Go to the last frame when the animation is interrupted
        COMPLETE,        // Complete the animation normally when interrupted
        COMPLETE_FAST,   // Complete the animation quickly when interrupted
        HOLD,            // Hold the current frame when the animation is interrupted
        REVERSE          // Reverse the animation when interrupted
    };

    AnimationFinishBehavior animationFinishBehavior;
    double animation_duration; // Duration of the animation in seconds
```

```
    double (*animation_easing_function)(double); // Easing function for the animation
};
```

and ChStyle perhaps:

```
template <class T>
struct ChStyle {
    // ...
    std::vector<T> animationStates;
    double animationDuration;
    EasingFunction easingFunction;
    // ...
};
```

obviously the drawing functions like ChPaint, ChPaintEdge, and ChPaintBody would need to incorporate the animation properties.
The BaseAnimation class could still handle animation logic and calculate the intermediate values

certainly the case of handling Mouse event (e.g., OnMouseEnter, OnMouseLeave)
would need to have some form of update to the animation system

```
void Button::OnMouseEnter() {
    // notify Update the animation state
    // ...

    // notify Start the animation
    // ...
}

void Button::OnMouseLeave() {
    // notify Update the animation state
    // ...

    // notify Start the animation
    // ...
}
```

Subject: Re: styling of widgets (animation / look and feel)
Posted by dodobar on Thu, 20 Apr 2023 21:55:20 GMT
View Forum Message <> Reply to Message

this code seriously lacks commenting :roll:

Subject: Re: styling of widgets (animation / look and feel)
Posted by mirek on Fri, 21 Apr 2023 05:56:06 GMT
View Forum Message <> Reply to Message

dodobar wrote on Thu, 20 April 2023 22:36agreed this was to get the ball rolling:

perhaps:

```
struct Style : ChStyle<Style> {
    Value look[4];
    Color monocolor[4], textcolor[4];
    Point pressoffset;
    int   focusmargin;
    int   overpaint;
    Font  font;
    Image ok, cancel, exit;
    bool  transparent;
    bool  focus_use_ok;

    // Animation properties
  enum AnimationFinishBehavior {
        GOTO_FIRST,      // Go to the first frame when the animation is interrupted
        GOTO_LAST,       // Go to the last frame when the animation is interrupted
        COMPLETE,        // Complete the animation normally when interrupted
        COMPLETE_FAST,   // Complete the animation quickly when interrupted
        HOLD,            // Hold the current frame when the animation is interrupted
        REVERSE          // Reverse the animation when interrupted
    };

    AnimationFinishBehavior animationFinishBehavior;
    double animation_duration; // Duration of the animation in seconds
    double (*animation_easing_function)(double); // Easing function for the animation
};
```

and ChStyle perhaps:

```
template <class T>
struct ChStyle {
    // ...
    std::vector<T> animationStates;
    double animationDuration;
    EasingFunction easingFunction;
    // ...
};
```

obviously the drawing functions like ChPaint, ChPaintEdge, and ChPaintBody would need to incorporate the animation properties.
The BaseAnimation class could still handle animation logic and calculate the intermediate values

certainly the case of handling Mouse event (e.g., OnMouseEnter, OnMouseLeave)
would need to have some form of update to the animation system

```
void Button::OnMouseEnter() {
   // notify Update the animation state
   // ...

   // notify Start the animation
   // ...
}

void Button::OnMouseLeave() {
   // notify Update the animation state
   // ...

   // notify Start the animation
   // ...
}
```

That is not a good design - too much for Button code to do.

However, meanwhile I found an elegant solution. Basically the only thing really needed for the style implementing code to allow it to perform animations is a pointer to widget as another ChPaint / ChLookFn parameter. If you have that, animation engine can hook into process easily (with special Values for look, can keep track of status with those, can even post time callbacks to refresh the look).

Moreover, we can make ChPaint a Ctrl method, which solves the problem without changing a single line of code in CtrlLib.

If this all sounds a bit confusing, do not despair. I will try to implement it over weekend or soon, together with reference example.

Mirek

Subject: Re: styling of widgets (animation / look and feel)
Posted by dodobar on Fri, 21 Apr 2023 09:40:54 GMT
View Forum Message <> Reply to Message

Thanks,
Yes I did not like the widgets having to pass back information to the style but I suppose 2 brains are better than one :roll:
baring any dependency issues the solution is nice and simple approach, I assume this is what you mean.

```
/**
@brief Structure representing a style in the GUI toolkit.
This structure encapsulates the visual and animation properties of a style
for use in the GUI toolkit. It includes values for look, colors, font, images,
and animation properties such as duration and easing function. By using this
structure, the look and feel of various widgets in the toolkit can be easily
customized and animated.
*/
template <class T>
struct ChStyle {
    // ...
    std::vector<T> animationStates;
    double animationDuration;
    EasingFunction easingFunction;
    Ctrl* widget; // Pointer to the widget this style is applied to
    // ...
};

struct Style : ChStyle<Style> {
    Value look[4];
    Color monocolor[4], textcolor[4];
    Point pressoffset;
    int   focusmargin;
    int   overpaint;
    Font  font;
    Image ok, cancel, exit;
    bool  transparent;
    bool  focus_use_ok;

    // Animation properties
    enum AnimationFinishBehavior {
        GOTO_FIRST,      // Go to the first frame when the animation is interrupted
        GOTO_LAST,       // Go to the last frame when the animation is interrupted
        COMPLETE,        // Complete the animation normally when interrupted
        COMPLETE_FAST,   // Complete the animation quickly when interrupted
        HOLD,            // Hold the current frame when the animation is interrupted
        REVERSE          // Reverse the animation when interrupted
    };

    AnimationFinishBehavior animationFinishBehavior;
    double animation_duration; // Duration of the animation in seconds
    double (*animation_easing_function)(double); // Easing function for the animation
};
```

One potential issue with adding the widget pointer to the Style struct is that it may not provide

enough control or understanding for some widgets. For example, if a widget wants to animate a specific area of its display that is not accounted for in the Style struct, it may not be able to do so easily.
at some level though the designer will need to set up the Style how he wants the animation/design.

looking at the base ChStyle, I'm not sure why you are using three specific images (naming) in a style "Image ok, cancel, exit;"
I can imagine you could have a MainImage being the master image/Icon used and an additional SubImage potentially covering the case of an icon with another icon representing a menu drop-down or something and if you really feel like it a userImage .
Some additional notes:
might be good to have an additional colour to represent border and text background.
also Gradents (how is this handled)

```
/**
@brief Structure representing a style in the GUI toolkit.
This structure encapsulates the visual and animation properties of a style
for use in the GUI toolkit. It includes values for look, colors, font, images,
and animation properties such as duration and easing function. By using this
structure, the look and feel of various widgets in the toolkit can be easily
customized and animated.
*/
struct Style : ChStyle<Style> {
    Value look[4];
    //Color monocolor[4], textcolor[4];
    Point pressoffset;
    int   focusmargin;
    int   overpaint;
    Font  font;
    //Image ok, cancel, exit;
    bool  transparent;
    bool  focus_use_ok;

    // Adjusted properties for images and colors (not sure if you need the array of 4 this might be for states)
    Image MainImage, SubImage, UserImage;
    Color textcolor[4], textBackground[4];
    Color bordercolour[4], foregroundcolor[4], altforegroundcolor[4] backgroundcolor[4],altbackgroundcolor[4];
    //EDIT: monocolor should just be desaturated of the main and not needed
    // Animation properties
    enum AnimationFinishBehavior {
        GOTO_FIRST,      // Go to the first frame when the animation is interrupted
        GOTO_LAST,       // Go to the last frame when the animation is interrupted
        COMPLETE,        // Complete the animation normally when interrupted
```

```
        COMPLETE_FAST,   // Complete the animation quickly when interrupted
        HOLD,            // Hold the current frame when the animation is interrupted
        REVERSE          // Reverse the animation when interrupted
    };

    AnimationFinishBehavior animationFinishBehavior;
    double animation_duration; // Duration of the animation in seconds
    double (*animation_easing_function)(double); // Easing function for the animation
};
```

Subject: Re: styling of widgets (animation / look and feel)
Posted by mirek on Sat, 22 Apr 2023 12:44:26 GMT
View Forum Message <> Reply to Message

Check this experimental branch

https://github.com/ultimatepp/ultimatepp/tree/animations

- the example is reference/animation (really simple, but as proof of concept should be fine)

Subject: Re: styling of widgets (animation / look and feel)
Posted by dodobar on Sun, 23 Apr 2023 14:27:29 GMT
View Forum Message <> Reply to Message

tks for the effort,
I would need more information on getting a new branch to run as this is new to me.

Is this just a case of downloading the branch theIDE and running or compiling the whole lib ?

cheers

Subject: Re: styling of widgets (animation / look and feel)
Posted by mirek on Mon, 24 Apr 2023 07:23:42 GMT
View Forum Message <> Reply to Message

https://www.ultimatepp.org/appidePackagesAssembliesAndNest s$en-us.html

---